Charter Data Checklist

This document provides a checklist for collecting and processing admissions lottery data from charter schools. It highlights the most common issues in cleaning lottery data. It is not exhaustive, and there is **no substitute** for perusing the data and making sure the code does what it's supposed to.

Code Hygiene

- ☐ **Create a makefile** (a code file to support codebase set up) that contains all the necessary packages and sets file paths for the project.
- ☐ **Add a descriptive header to the top of each code file** that documents the purpose of the code, which datasets are inputs, and which datasets are outputs.
- ☐ **Set up an intuitive, structured folder system** for code, intermediary datasets, and output.
    - Create separate code files for each school and cleaning stage. Organize the school's data processing files into a subdirectory.
        - Use consistent naming practices in your code file names. Code that covers the same cleaning step should be named similarly.
    - For the sake of replicability, create folders programmatically rather than manually (e.g., using the mkdir command in Stata).
- ☐ **Utilize master do-files**
    - Consider creating a master code file for each school that contains switches for each code file in the data processing pipeline. This way, you can easily control which sections of the cleaning and matching process you would like to run for a given school.
    - Consider creating an overarching master code file that contains switches for each school-level code file. This way, you can run code for multiple schools at the same time.
- ☐ **Save an intermediate dataset** at the end of each section turned on by switches. This way, you can run individual sections without having to rerun all the preceding sections.
- ☐ **Label each dataset** so that you know its purpose. It can be hard to infer from the filename alone, especially when you may have more than a dozen intermediate files with somewhat similar names.
    - We recommend including the school, years covered, and data processing stage in each dataset's name.
- ☐ **Use assert commands frequently** to check your work!
- ☐ **Write programs for complicated and frequently-used processes**, such as geocoding.

Initial Data Processing Steps

Importing Datasets

- ☐ **Always import the raw datasets with code** (e.g., *import excel* or *import delimited* in Stata). Avoid any manual steps for importing datasets.
- ☐ **Import all columns as string variables to make merging and appending easier**. You can convert variables to numeric and date types in the cleaning stage.
- ☐ **Be careful when manipulating the original raw file**. Never replace the original file with a version with saved changes. Consider creating a backup raw file.

- ☐ **Label each file sensibly.** One option is creating a variable denoting the file name (i.e., *gen year = `i',* if the file contains lottery data for year `i'*)* so that you can label the files for different years programmatically.

Merging Datasets

Sometimes, you may need to merge different datasets together to get all the relevant lottery information in one year. For example, with SchoolMint data, it is often necessary to merge an application report with a custom report containing every applicant's lottery number.

- ☐ **Check that your merging variable uniquely identifies observations.**
- ☐ **Note how many observations are not found in both datasets** and iterate with your data providers to address missing data as necessary.

Appending Datasets

You will also need to append datasets to create one working lottery dataset. For example, if each file is associated with a year of lottery data, you will want to append these together to construct your full dataset.

- ☐ **Before appending yearly files, verify that variable names are consistent across all files** (e.g., the variable denoting the student's first name might be called "First Name" in the 2016 file, but "first_name" in the 2017 file. The names should be made consistent across years/files. Otherwise, you might end up with missing values of the first name variable!)
- ☐ **Label the appended file.**

Clean Student Names

Cleaning student names consistently across years and schools is crucial because names will be used in the fuzzy-matching process to match charter lottery applicants to their enrollment records.

- ☐ **Strip off:**
    - Combining marks (e.g., "-" in Smith-Jones)
    - Accents (e.g., umlauts)
    - Special characters
    - Suffixes (e.g., Jr., II, III)
    - All beginning, trailing, and multiple consecutive spaces
- ☐ **Convert the name variables either to all lowercase or all uppercase**. Whatever you do, be consistent across years and schools!
- ☐ **Standardize how you flag students with no middle name**. Typically we replace other values indicating a missing middle name with "NMN."
    - Some common indicators that a student does not have a middle name: NA, NONE, NM, NO, NOMIDDLE, NOMIDDLENAME, NONAME, NOTAPPLICABLE.
    - During the fuzzy matching process, you may run into many different values that indicate a student has no middle name, causing students not to match across

datasets. For example, a student may be listed as “John N/A Smith" in one dataset and “John DOESNTHAVEAMIDDLENAME Smith" in another. You can manually mark these as matches in the fuzzy matching step, but we suggest going back to the cleaning script and adding to the list of values to replace with NMN.

Clean Dates

The lottery data will contain date variables, such as the date a student submitted their application (“submission date”), the date a student received an offer from a charter school (“offer date”), and the student’s birth date.

These dates are crucial for the analysis. Submission dates allow us to infer which applications were submitted late and therefore were not included in the lottery. Offer dates allow us to infer which students received an offer. Student birth dates are used to match student application data to administrative student records.

- ☐ **Get rid of any extraneous text** (e.g., timezones EDT or EST) that may get in the way of converting dates from strings to doubles.
- ☐ **Detect different date formats across files and write code to handle them**. Sometimes date variables are formatted differently by year of the lottery file.
- ☐ **Add an `assert` command that checks that dates were cleaned properly** and no additional missing values were generated in the process.
    - To accomplish this, you could to save the raw, uncleaned variables and generate a cleaned version to compare.

Coding Priority Groups

Correctly coded priority groups allow us to control for every applicant’s “risk” (i.e., probability) of being offered a charter school seat. For example, an applicant whose sibling attends the school they are applying to will have a different risk of an offer than an applicant who just lives in the public school district. Some things to look out for when coding the priority groups:

- ☐ **Locate and refer to documentation confirming the priority group rules.**
- ☐ The lottery datasets usually have some string variable with text stating what type of priority groups there are. **Inspect all the possible values of this string variable and determine how you will recode them to a numbered priority group**.
    - For example:
        - priority_group = 1 for siblings of current students
        - priority_group = 2 for in-zone applicants
        - priority_group = 3 for out-of-zone applicants
    - The `levelsof` command in Stata is helpful to determine the possible values of the priority group string variable.
- ☐ **Generate a binary variable for each priority group**
    - In Stata, `strpos` is helpful here if your dataset has a string variable containing all the possible priority groups.

        - Ex: gen byte sibling_attending = strpos(priorities_string, “Sibling Attending”) > 0
- ☐ **Verify that your categorizations of the string priority group variable to the recoded priority group variable are mutually exclusive and collectively exhaustive**. In other words, you want to make sure that you are not overwriting previously recoded applicants (mutually exclusive), and you also want to make sure you’re recoding priority groups for all applicants (collectively exhaustive)! This is a useful place to use assert commands.

Coding Offer Variables

- There are two strategies for coding initial offers:
    - Option 1: Use documentation from the school or from the Wayback machine to find the lottery date. Students with that offered_date can be coded as initial_offer == 1. You should try this option first.
    - Option 2: Tabulate the offered_date lottery and find the earliest offered_date. Students with the earliest offered_date can be coded as initial_offer == 1.
        - Sometimes, the earliest offered_date used in Option 2 differs from the actual lottery date used in Option 1. In that case, you should use Option 2.
- **Use an `assert` command to check that ever_offer is only switched on when initial_offer is switched on.**

Invalid Applications

The lottery datasets often include observations (applications) that are considered invalid for various reasons. Cases of invalid applications include:

- Late applications: students who submitted their applications after the application deadline and are therefore not included in the lottery.
- Applications with either missing or invalid names or dates of birth.
- Applications with missing or implausible (e.g., negative) lottery numbers.
- Applications with implausible birthdates (e.g., birthdates outside the +/-3 range of the mode).
- Applications with missing school, grade, or year variables.
- Canceled applications.

- ☐ **Flag these different invalid applications** so that they can be identified easily in later stages of the analysis.
- ☐ **Consult with the lottery documentation or contacts at the charter schools** to determine which types of applications are considered invalid.

Duplicates

- ☐ **Consider creating a unique ID variable before dropping any observations** (see the fuzzy matching section).

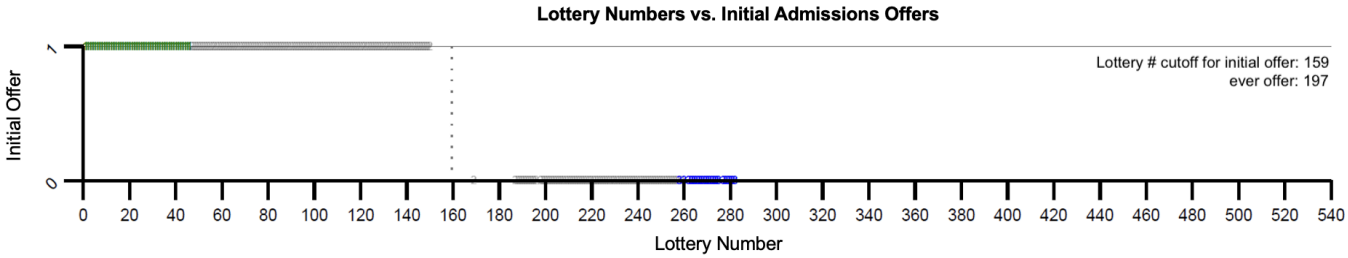
- ☐ **Genuine duplicates to flag**: different students who applied to the same lottery (school-grade-year) and received the same lottery number, the same student who applied to the same lottery (school-grade-year) multiple times and received multiple lottery numbers.
- ☐ **Duplicate observations to drop**: the same student who applied to the same lottery (school-grade-year) and received the same lottery number.
- ☐ For charter management organizations (CMOs) with multiple campuses, it sometimes appears that students applied to multiple campuses. In reality, they applied to the CMO as a whole, but the school adds another observation when they get an offer to a specific campus. In these cases, offer variables should be coded based on the second observation created.

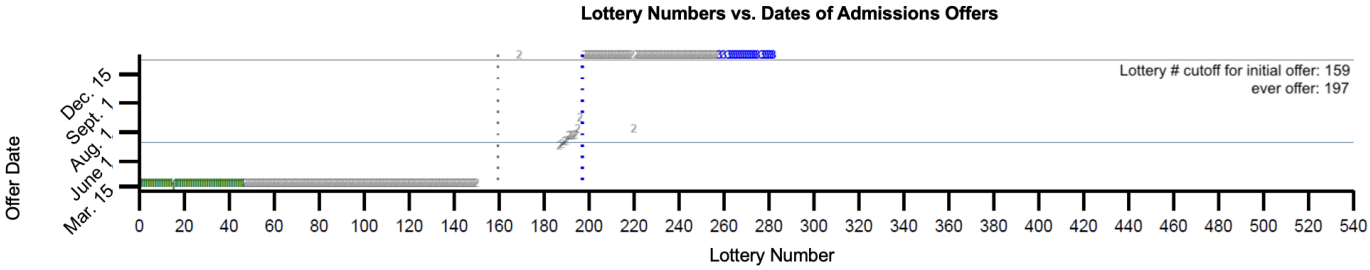
Making Sense of Lottery Numbers

- ☐ **Check whether the offered dates, lottery numbers, and priority groups are consistent** with each other through visual plots.

The most helpful versions plot initial_offer and offered_dates on the y-axis and lottery numbers on the x-axis, with priority groups color-coding each observation. Examples of these plots are given below. These plots can help you quickly determine if something fishy is going on. A plot is created for every year-grade-school. For example:

- Are offers being given “out-of-order”? For example:
    - Students with priority group 3 are getting offers before students in priority group 1
- Are priority groups inconsistent with lottery numbers? I.e., as you go from lottery number 1 to N, does the priority group progress from 1, to 2, to 3, or does it jump back and forth inconsistently?





The best way to use these plots is to pick a year-grade-school where something is “off” and then inspect the data directly to see what went wrong. Was it the coding of the offer variable, priority groups, etc., that caused things to go wrong? Or is something wrong with the raw data itself?

Making Sense of the Lottery

- ☐ **Check that applicant characteristics are balanced** across offered and non-offered applicants.

Theoretically, the applicants offered a seat and those not offered a seat through the lottery should share the same characteristics (on average). We check this by conducting balance checks for applicant characteristics by offer status:

- These are conducted at the school and grade level (typically the major entry grades: 5th or 6th grade and 9th grade) on the set of valid applications.
- Key demographics include gender, race, SPED status, FRPL status, age (in months), in-district at baseline, and neighborhood characteristics where available.
- Relevant offer statuses include initial offer and ever offer.
- Regress each characteristic on offer status controlling for risk set (the likelihood an applicant will receive an offer).

Imbalance at the school level need not cause concern. It’s likely that the pooled school sample will be balanced even if some schools’ lottery winners and losers are imbalanced. If desired, you can explore imbalance for a given school and grade further by subsetting into risk sets to identify the risk set(s) where imbalance occurs.

Fuzzy Matching

- ☐ Matching lottery records to state administrative records may require fuzzy matching. One great option is using the reclink2 command in Stata with name (first, middle, and last) and date of birth as the merging criterion.[1]
- ☐ **Make sure student names and birthdates are properly and consistently cleaned and formatted!**

[1] Several recent papers have merged on name and date of birth, including Rose et al. (2022), Norris et al. (2021), and Baron et al. (2022).

    - Consider using phonetic algorithms such as Stata's soundex or nysiis commands.
- ☐ **Use a timer to measure run times and document the output for future reference and planning**. Fuzzy matching commands tend to have long run times.
- ☐ **reclink2 requires a unique ID in the master dataframe (the lottery data) and the using dataframe (administrative data)**. You will likely have to generate this ID yourself.
    - Consider generating the unique ID before any applications are dropped so the numbering is not sensitive to keep/drop coding decisions.
    - Consider concatenating the year with an id_counter so the addition of new/updated yearly data does not change the unique IDs of previous observations.
    - Add checks to confirm that your unique ID is truly unique.
- ☐ **Rerun the fuzzy matching code only when necessary**. The matching process is lengthy and can be tedious. It may be necessary to rerun when:
    - New lottery data is added (e.g., a new year of data is added or a new version of the 2022 lottery data is added to replace the old).
    - New years of student administrative records become available.
- ☐ After running the fuzzy match algorithm, **sort the observations by fuzzy matching score and determine an arbitrary cut-off score at or above which all observations will be automatically marked as matches.** This cutoff will vary by the dataset. At Blueprint, we've used cutoffs between 0.7 and 0.85.
- ☐ **Implement rules of thumb for observations below the cutoff to mark as automatic matches** to improve replicability and reduce the number of observations that must be manually matched.
    - Some string variables that are helpful to create for this:
        - FirstName + MiddleName + LastName
        - MiddleName + LastName
        - FirstName + MiddleName
    - Some examples of these rules may include:
        - Master and Using have the same last name and same birthdate but swapped first and middle names.
            - Ex: JOSHUA JACK JOHNSON and JACK JOSHUA JOHNSON
        - Master and Using have the same first name and same birthdate but combined middle name and last names.
            - Ex: JOSHUA JACK JOHNSON and JOSHUA JACKJOHNSON
        - Master and Using have the same first name and birthdate but swapped middle name and last name.
            - Ex: JOSHUA JACK JOHNSON and JOSHUA JOHNSON JACK
        - Master and Using have the same last name and birthdate but combined middle name and first name.
            - Ex: JOSHUA JACK JOHNSON and JOSHUAJACK JOHNSON
        - Master and Using have the same first name, last name, and birthdate, but middle name is missing from one dataset.
            - Ex: JOSHUA JACK JOHNSON and JOSHUA JOHNSON

        - Master and Using have the same first name, last name, and birthdate, but only a middle initial is used in one dataset.
            - Ex: JOSHUA JACK JOHNSON and JOSHUA J JOHNSON
    - **Keep an eye out for Hispanic naming conventions.** Students may have multiple last names listed in the administrative dataset and only one last name listed in the lottery data.
        - Use `strpos` in Stata to identify these cases and mark them as matches. Ex: strpos(using_last_name, master_last_name) > 0.
- ☐ When determining if a fuzzy match below the cutoff is a true match, it's helpful to compare the grade to which the student in the lottery data is applying and the grade in which the student in the administrative data is currently enrolled. Generally, grade_current + 1 should equal grade_applying.
    - For example, if a student is currently enrolled in grade 5, it's unlikely that they are applying for grade 3.
    - For cases in which the grade_current + 1 does not equal grade_applying, employ more stringent rules when marking manual matches (i.e., names have to be nearly identical).
- ☐ After running the fuzzy matching process using the criteria of both name (first, middle, last) and date of birth, **consider fuzzy matching just on name as a second step**.
    - There may be cases in which parents mistakenly enter the wrong birthdate. Common instances of this include switched days and months (i.e., 06/10/2004 and 10/06/2004), switched digits (i.e., 06/10/2004 and 06/01/2004), and wrong months (06/10/2004) and (07/10/2004).
    - Generating a numeric variable that captures the difference between the two birth dates is helpful here. For example, we generally consider all proposed matches in which the birthdates are more than 3 years apart to be false matches.
        - This can also help identify the common instances of misinput birthdays. For example, birth dates 30 or 365 days apart may be more likely to be true matches.
    - When excluding date of birth as a fuzzy matching criterion, be less lenient about the differences in names that you'll consider matches. For example, you may require names to match exactly or almost exactly.

Some additional notes on reclink2:

- Reclink2 creates duplicate observations when two or more observations in the using dataset match to an observation in the master dataset with **exactly the same score**. This normally happens with scores well below the cutoff that separates "true" matches from "false" matches (more on this under *"manual review"* below), but for the few cases where it happens above the cutoff, manual review should help decide which match to keep. For the remaining duplicates, use `duplicates drop` after each iteration of reclink2.
- Command options:
    - **required**: When using required(var1 var2) the variables var1 and var2 must match exactly for the observations to be considered a match. When matching

lottery data to enrollment files, it can help to match on year and student date of birth.
    - **exactstr**: This option is useful when matching with a numerical variable and that variable is not under the option *required*. Since reclink2 uses a string comparator, the dob 7/28/2000 would be considered a closer match to 7/28/2009 than the dob 7/28/1999, even though the latter is a more reasonable match. This option overcomes this problem since the variable is compared with a 0/1 agreement rather than with a degree of similarity given by the string comparator.
    - **uvarlist**: Use this option if the master and using files have different names for any of the matching variables. The using variable will still be renamed to match the name in the master dataset, but it avoids the extra step of having to rename the variables before being able to use reclink2.
    - **manytoone**: Use this option if there is any possibility that there are any duplicates/multiple application applicants (either perfect or with differences in spelling) in the master dataset. This way both (or more) records for the same applicant are matched to one observation in the using dataset. This doesn't avoid the issue that there shouldn't be any duplicates in the using dataset. This option is not available in reclink, hence we use reclink2.
    - **npairs**: This option allows you to keep multiple potential matches rather than only the match with the highest score (default). In Blueprint's experience, when we tried using it to match lottery data to enrollment data, there were no cases where observations had more than one (reasonable) potential match (or if there was more than one, the match with the highest score was always the correct one). So, in this case, it was not necessary, but it may be useful if there is the possibility that there are duplicates (with spelling differences) in the using dataset. This option is not available in reclink.

Additional Resources

- Innovations for Poverty Action's page on fuzzy matching
- Abramitzy et al.'s guidance on linking historical data and examples of different algorithms that can be implemented
- NBER Reporter article on linking historical census data
- Working with date-time variables in Stata is tricky. These two documents (A and B) give an overview of how to manipulate date-time data. For example, they describe how to convert strings containing date/time information into doubles with date/time formatting.